We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

1

Automated Generation of User Interfaces – A Comparison of Models and Future Prospects

Helmut Horacek, Roman Popp and David Raneburger
Institute of Computer Technology, Technical University of Vienna
Austria

1. Introduction

In the past decade, demands on interfaces for human-computer interaction (HCI) as well as efforts invested in building these components of software systems have increased substantially. This development has essentially two sources: Existing tools do not well support the designer, so that building these components is time-consuming, error-prone, and requires substantial programming skills. Moreover, the increasing variety of devices with different presentation profiles, variations on media uses and combinations of several media points to a necessity of designing some sort of interface shells so that one such shell can be adapted to a set of partially divergent needs of varying presentation forms.

Especially the second factor, as also argued by Meixner & Seissler (2011), makes it advisable to specify interfaces on some sort of *abstract* level, from which operational code can be generated automatically, or at least in some semi-automated way. This aim is quite in contrast to traditional, mostly syntactic specification levels. Abstract level interfaces should not only be better understandable, especially by non-programmers, but they would also allow for a systematic adaptation to varying presentation demands, as advocated for above. Apart from the ambitious goal to define an appropriate design language and tools for building interfaces in this language, a major difficulty with such models lies in the operationalization of specifications built on the basis of these models, both in terms of degrees of automation and in terms of quality of the resulting interface appearance and functionality. Since semantic interaction specifications can abstract away plenty of details that need to be worked out for building a running system, we can expect that there is a fundamental tension between ease and intuitiveness of the design on the one hand, and coverage and usage quality of the resulting interface on the other hand.

To date, a limited set of development models for interface design have been proposed, which are in line with the motivations as outlined above: discourse-based communication models (Falb et al. (2006)), task models (Paternò et al. (1997), Limbourg & Vanderdonckt (2003)), and models in the OO method (Pastor et al. (2008)). Moreover, abstract models of interface design bear some similarities to natural language dialog systems and techniques underlying their response facilities, including reasoning about content specifications based on forces of underlying dialog concepts, as well as measures to achieve conformance to requirements of form. Therefore, we elaborate some essential, relevant properties of natural language dialog systems, which help us to develop a catalog of desirable properties of abstract models for interface design. In order to assess achievements and prospects of abstract models for

interface design, we compare some of the leading approaches. We elaborate their relative strengths and weaknesses, in terms of differences across models, and we discuss to what extent they can or cannot fulfill factors we consider relevant for a successful interface design. Based on this comparison, we characterize the current position of state-of-the-art systems on a road map to building competitive interfaces based on abstract specifications.

This paper is organized as follows. We first introduce models of natural language dialog systems, from the perspective of their relevance for designing HCI components. Then we present a catalog of criteria that models for designing interfaces should fulfill to a certain extent, in order to exhibit a degree of quality competitive to traditionally built interfaces. In the main sections, we present some of the leading models for designing interfaces on abstract levels, including assessments as to what extent they fulfill the criteria from this catalog. Next, we summarize these assessments, in terms of relative strengths and weaknesses of these models, and in terms of where models in general are competent or fall short. We conclude by discussing future prospects.

2. Linguistic models

Two categories of linguistic models bear relevance for the purposes of handling discourse issues within HCIs:

- Methods for *dialog modeling*, notably those based on information states. This is the modern approach to dialog modeling that has significantly improved the capabilities of dialog systems in comparison to traditional approaches, which are based on explicit, but generally too rigid dialog grammars.
- Methods for *natural language generation*, which cover major factors in the process of expressing abstract specifications in adequate surface forms. They comprise techniques to concretize possibly quite abstract specifications, putting this content material in an adequate structure and order, choosing adequate lexical items to express these specifications in the target language, and composing these items according to the constraints of the language.

Apparently, major simplifications can be made prior to elaborating relations to the task of building HCIs: no interpretation of linguistic content and form is needed, and ambiguities about the scope of newly presented information also do not exist. Nevertheless, we will see that there are a variety of concepts relevant to HCIs, which makes it quite worth to study potential correspondences and relations.

Dialog models with information states have been introduced by Traum & Larsson (2003). According to them, the purpose of this method includes the following functionalities:

- updating the dialog context on the basis of interpreted utterances
- providing context-dependent expectations for interpreting observed signals
- interfacing with task processing, to coordinate dialog and non-dialog behavior and reasoning
- deciding what content to express next and when to express it

When it comes down to more details, there are not many standards about the information state, and its use for acting as a system in a conversation needs to be elaborated – recent approaches try to employ empirically based learning methods, such as Heeman (2007).

Semantically motivated approaches typically address certain text sorts or phenomena such as some classes of speech acts, in abstract semantics. Elaborations have been made for typical situations in information-seeking and task-oriented dialogs, including grounding and obligations, such as Matheson et al. (2000), and Kreutel & Matheson (2003). Altogether, information state-based techniques regulate locally possible dialog continuations, as well as some overarching contextual factors.

For purposes of HCI development, a few of these underlying concepts pertain:

- Sets of interaction types that regulate the coherence of the discourse continuation in dependency of the category of the immediately preceding interaction. For instance, questions must normally be answered, and requests confirmed, prior to executing an action that satisfies the request.
- Changes in the joint knowledge of the conversants according to the state of the discourse (*grounding*). For example, specifications made about properties of a discourse object should be maintained – e.g., an article to be selected eventually, as long as the interaction remains within the scope of the task to which this discourse object is associated.
- Holding evident *commitments* introduced in the course of the interaction, which essentially means that a communicative action that requires a reaction of some sort from the other conversant must eventually be addressed unless the force of this action is canceled through another communicative action. For example, a user is expected to answer a set of questions displayed by a GUI to proceed normally in this dialog, unless he decides to change the course of actions by clicking a 'back' or 'home' button or he chooses another topic in the application which terminates the subdialog to which the set of questions belongs.

The other category of linguistic models, methods for natural language generation, are characterized by a stratified architecture, especially used in application-oriented approaches (see Reiter (1994)). There are three phases, concerned with issues of *what* to say, *when* and *how* to say it, mediating between four strata:

1. A *communicative intention* constitutes the first stratum, which consists of some sort of abstract, typically non-linguistic specifications. Through the first phase called *text planning*, which comprises selecting and organizing content specifications that implement the communicative intention,
2. a *text plan*, the second stratum is built. This representation level is conceived as *language-independent*. Through operations that fall in the second phase, including the choice of lexical items and building referring expressions,
3. a *functional description* of some sort, the third stratum, is built. This representation level is generally conceived as *form-independent*, that is, neither surface word forms nor their order is given at this stage. However, details of this representation level differ considerably according to the underlying linguistic theory. Through accessing information from grammar and lexicon knowledge sources
4. a *surface form* is built, which constitutes the fourth stratum, the final representation level.

Especially the criterion of language independence of the text plan is frequently challenged on theoretical grounds, since the desirable (and practically necessary) guarantee of *expressibility* (as argued by Meteer (1992)) demands knowledge about the available expressive means in the target language. The repertoire of available linguistic means bears some influence on how content specifications may or may not be structured prior to expressing them lexically. Since

transformations are typically defined in an easily manageable, widely structure-preserving manner, a high degree of structural similarity across representations from adjacent strata is essential. In order to address this problem in a principled manner, several proposals with interactive architectures have been made, to enable a text planner to revise some of its tentative choices, on the basis of results reported by later phases of processing. These approaches, however, were all computationally expensive and hard to control. In practical systems, a clever design of concrete operations on text planning and subsequent levels of processing, as well as care with the ontological design of the text plan level stratum proved to be sufficient to circumvent problems of expressibility.

It is quite remarkable, that these four strata in architectural models of natural language generation have a strong correspondence in the area of GUI development, in terms of Model Driven Approaches. In both models, higher level strata are increasingly independent of properties of the categories of proper expressive means, which are language and form in the case of natural language, and platform and code in the case of GUI development. The connection between these models becomes even tighter when we take into account multi-modal extensions to natural language generation approaches, where components in a text plan can be realized either by textual or by graphical means, including their coordination.

When it comes to examining the relevance of concrete methods originating from natural language generation for HCI purposes, several measures offer themselves, which are all neutral with respect to the proper features of natural language:

- Techniques for organizing bits and pieces of content in ontological and structural terms, following concepts of coherence, as encapsulated in a number of theories, such as Rhetorical Structure Theory, see Mann & Thompson (1988). Dominating relations on this level are hierarchical dependencies, while form and order are expressed in terms of constraints, which come to play not before concrete realizations are chosen.
- Choices between expressive means, primarily between alternative media, according to their suitability to express certain categories of content. For example, causal relations or negation elements can be presented much better in a textual rather than in a graphical form, whereas the opposite is the case for local relations.
- Structural and ontological relations may also drive the suitability of form and layout design. For example, groupings of items need to be presented in a uniform, aligned manner. Moreover, background information should be presented in moderately salient forms, quite in contrast to warnings and alert messages.

In addition, it is conceived that automated approaches to natural language generation are generally good in producing texts that are conform to norms of several sorts, such as the use of a specific vocabulary and limited syntactic forms, but also non lexically dependent conventions.

In the following sections, we refer to various aspects of linguistic models, when comparisons between models of GUI construction and transformations between representation levels are discussed.

3. Criteria

The goal of building interfaces on some level of abstract specifications is ambitious, and implementations of conceptual approaches comprise a variety of measures and the adequate orchestration of their ingredients. Consequently, assessments made about competing

approaches can be broken down into a set of dimensions, where elaborations in individual approaches can be expected to address some of these dimensions in partially compensative degrees. Within this section, we apply the term 'user' to refer to an essentially untrained person who *uses* such an approach to develop an interface.

As for any software system to be built automatically or at least semi-automatically on the basis of abstract, user-provided specifications, three orthogonal criteria offer themselves:

- The *ease of use*,
  that is, the amount of training needed, prior knowledge required, and degree of effort demanded to generate adequate specifications for some piece of application.
- The *degree of operationalization*,
  that is, where the position of an approach resides on the typically long scale ranging from moderately semi-automated to fully-automated systems.
- The *coverage*,
  that is, to what extent and in what ways an approach can bring about the ingredients needed for the system to be built, hence, what kind of situations it can handle and for which ones it falls short for some reason.

In addition to these in some sense basic criteria, there are two further ones, which go beyond the development of a single system in complementary ways:

- *Adaptability in the realization*,
  that is, using the system ultimately generated in different contexts, thereby taking into account specific needs of each of these contexts, and making use of a large portion of the specifications in all contexts considered.
- *Reuse of (partial) specifications*,
  that is, the use of specifications or of some of their parts in several components of a model specified by an approach, or across different versions.

In the following, we flesh out these criteria for the specific task at hand.

As for the *ease of use*, the user should be discharged of technical details of interface development as much as possible. Ideally, the user does not need to have any technical experience in building interfaces, and only some limited teaching is required, so that the user becomes acquainted with operations the development system offers and with the conventions it adopts. In order to make this possible, the implementation of an interface development model should foresee some language that provides the building blocks of the model, and effective ways to compose them. In addition to that, certain features aiming at the support in maintaining correctness and/or completeness of specifications made can prove quite useful. While correctness proofs for programs are expensive and carried out for safety-critical tasks, if at all, measures to check completeness or correctness in some local context are much easier to realize, and they can still prove quite valuable. For example, the system might remind the user of missing specifications for some of the possible dialog continuations, according to contextually suitable combinations of communicative acts. We have filed these measures under the item *ease of use* because they primarily support a user in verifying completion and correcting errors of specifications if pointed to them, although these features can also be conceived as contributions to *degrees of operationalization*.

The *degree of operationalization* itself constitutes methods and procedures which regulate how the interpretation of a model specified by a user is transduced into executable modules, especially what activities involved in these procedures need to be carried out by hand. These measures manifest themselves in three components complementing each other:

- The *discourse structure*
- The *incorporation of references to business logic components*
- Invoking *rendering techniques*

The *discourse structure* per se constitutes the proper model which the user has to build in terms of abstract specifications. The major challenge from the perspective of the operationalization lies in providing an automated procedure for transducing the abstract specifications made into a workable system. Since setting up such a procedure is normally associated with plenty of details that go beyond of what is represented in the abstract specifications made by the user, it is particularly important to automate the derivation of all necessary details as much as possible.

The *incorporation of references to business logic components* is, strictly speaking, a subcategory of activities concerning specifications of the discourse structure. Since this particular activity is so prominent – it occurs in absolutely all models, in a significant number of instances, and is potentially associated with quite detailed specifications – we have given it a first class citizen state for our considerations. Moreover, handling this connection is also a primary task supported by the information state in linguistic models. As for linguistic models, it is generally assumed that the business logic underlying an application is properly and completely defined when interface specifications are to be made, in particular for establishing references to business logic components. However, when developing a software system, it is conceivable that some functionality originating from the discourse model may point to a demand on the business logic which has not been foreseen when this component has been designed; this situation is similar to the building of discourse models in computational linguistics, where discourse objects are introduced in the course of a conversation, which exist within the scope of this conversation only, and are related to, but not identical to some real world objects. For example, in a flight booking application, one has to distinguish between the proper flights in the database, completed flight specifications made by a customer built in the course of some customer-system subdialog, and partial, potentially inconsistent flight specifications incrementally made by the customer in the course of this dialog. Since it is generally unrealistic to assume perfect business logic design in all details, some sort of an interplay between the definition of the business logic and the design of the discourse structure may eventually be desirable. Finally, access to business logic components for reference purposes can also vary significantly in their *ease of use* across approaches, so that we have to consider this issue from the usability perspective as well.

Invoking *rendering techniques* is somehow converse to the other categories of handling specifications. It comprises how and where information can be specified that rendering methods additionally require in order to produce compositions of concrete interaction elements in an appropriate form. There are similarities between the role of rendering and components in the production of text out of internal specifications, as pursued in computational linguistics. The production of text comprises measures to assemble content specifications followed by methods to put these into an adequate linguistic form. Rendering techniques essentially have relations to the second part of this process. These techniques comprise mappings for the elements of the abstract specifications, transducing them into

elements of a GUI or of some other dedicated presentation device, as well as constraints on how the results of these mappings are to be composed to meet form requirements of the device addressed. The overall task is most suitably accomplished by automating mapping specifications and device constraints as much as possible, and by providing a search procedure that picks a mapping combination in accordance with the given constraints, thereby obeying preference criteria, if available. In most natural language generation system architectures, especially those of practical systems, locally optimal choices are made in a systematic order, thus featuring computational effectiveness and simplicity of control, at the cost of sacrificing some degree of potentially achievable quality. A few clever search procedures exist, improving that quality with limited extra effort. In an elaborate version, one can expect that this process is characterized by compensative effects between search effort and quality achievement. A useful property of automated rendering techniques, similar to some natural language generation applications, is the conformance to style conventions and preference constraints, which can be ensured by the automation of form choice and composition.

The *coverage* of a discourse model in terms of discourse situations addressed may vary significantly across individual approaches. For elaborate versions, a considerably large repertoire of discourse situations and their flexible handling can prove to be important, following the experience from natural language dialog systems. For these systems, much effort has been invested in expanding the kind of discourse situations covered, which proved to be valuable, since the increased flexibility improved the effectiveness of dialog task achievement considerably.

We distinguish discourse situations according to structural relations between components of such situations. The more involved these relations are, the more challenging is it to provide the user with tools to make abstract specifications of the underlying discourse situation in an effective manner. We consider the following situations, in ascending order of complexity:

- *Groupings*
  This structural pattern constitutes a limited set of items of the same kind, which have to be addressed in the same fashion. Unlike in human spoken dialogs, they can be treated in one go in many HCI devices, such as a GUI. A typical example is a pair of questions concerning source and destination of a trip, and the associated answers.
- *Embeddings, such as subdialogs*
  In many discourse situations, an elaboration of the item currently addressed may be required. This may concern supplementary information, such as a property of some airport chosen as destination, or, most frequently, a clarification dialog, asking, for example, to disambiguate between two airports that are in accordance with some specification made so far.
- *Conditional branching*
  The appropriate continuation in a discourse situation may depend on some specific condition that arose during the preceding course of the dialog, for example through unexpected or faulty specifications. In many cases, this condition manifests itself in the category of the immediately preceding utterance or of its content, such as an invalid date specification, but it may also be the value of some recently computed state variable, such as one which makes an incompatibility between a set of query specifications explicit. The continuation after the branching may completely diverge into independent continuations, or a subdialog may be started in one or several of these branches, after the completion of which control may return to the point where the branching is invoked.

- *Repetitions and related control patterns*
  In many situations, certain discourse patterns are invoked repeatedly, mostly in case of a failure to bring about the goal underlying the fragment which conforms to this pattern. Repetitions may be unlimited, if the human conversant is supposed to provide a suitable combination of specifications within this discourse fragment, and he can retry until he succeeds or he may decide to continue the dialog in some other way. Repetition may also be constrained, for example by a fixed number of trials, such as when filling out a login mask, or when specifying details of some payment action.
- *Simultaneous and parallel structures*
  Most dialogs simply evolve as sequences of utterances over time. In some situations, however, the proper dialog can reasonably continue in parallel to executing some time-consuming system action. One class of examples concerns processing of computationally heavy transactions, such as a database request, during which the proper dialog can continue, with the result of the database request being asynchronously reported when available. Another class of examples concerns the play of a video or of a slide show, which can be accompanied by a dialog local to the context where the video respectively slide show is displayed.
- *Topic shifts, including implicit subdialog closing*
  This kind of discourse situation is the most advanced one, and it can also be expected to be the most difficult one to handle. In human conversations, topic shifts are signaled by discourse cues, thereby implicitly closing discourse segments unrelated to the newly introduced topic, which makes these shifts concise and communicatively so effective. Within a GUI, similar situations exist. They comprise structurally controlled jumps into previous contexts, frequently implemented by *Back* and *Home/Start* keys, as well as explicit shifts to another topic which is out of the scope of the current discourse segment. An example is a customer request to enter a dialog about car rental, leaving a yet uncompleted dialog about booking a flight. As opposed to human dialogs, where the precise scope of the initiated subdialog with the new topic needs to be contextually inferred, these circumstances are precisely defined within a GUI. However, providing mechanisms for specifying these options in terms of abstract discourse specifications in an intuitive manner and with limited amount of effort appears to be very challenging.

Adaptability in the realization may concern a set of contextual constraints. One of them comprises specificities of the device used, such as the available screen size, which may be significantly different for a laptop and for a PDA. Another distinction lies in the use of media, if multiple media are available, or if versions for several ones are to be produced. For example, a warning must be rendered differently whether it comes within a GUI or whether it is to be expressed in speech. Finally, the ultimate appearance of an interface may be varied according to different conventions or styles.

Reuse of partial specifications also may concern a number of issues. To start with, partial or completed specifications of some discourse situation, including specifications for rendering, may be modified according to demands of other styles or conventions – the purpose is identical to the one described in the previous paragraph, but with a different timing and organization. Moreover, the incorporation of subdialog patterns is a very important feature, useful in some variants. One possible use is the provision of skeletons that cover subdialog patterns, so that they can be instantiated according to the present discourse situation. Another possible use is the reoccurrence of an already instantiated subdialog pattern, which may be reused in another context, possibly after some modifications or adaptations to the concrete

instantiations are made. Finally, versioning may be an issue, either to maintain several versions for different uses, or to keep them during the design phase, to explore the differences among them and to pick a preferred one later. Most of these reuses of partial specifications can be found in natural language generation systems, but this is hardly surprising, since almost all of them are fully automated systems.

This catalog of criteria is quite large, and some of the items in this catalog are quite advanced, so that few of the present approaches if any at all can be expected to address one or another of these advanced items, even to a limited degree. Most items in this catalog do not constitute black-or-white criteria, which makes assessing competing approaches along these criteria not an easy job. Moreover, approaches to design interfaces on some abstract specification level are not yet far enough developed and documented so that detailed, metric-based comparisons make sense. For example, the *ease of use*, in terms of the amount of details to be specified and the intuitiveness of use have to be assessed largely for each model separately, on the basis of its specificities, since experimental results about these user-related issues are largely missing. Altogether, we aim at a characterization of the current position of state-of-the-art systems, in terms of their relative strengths and weaknesses, as well as in terms of how far the state-of-the-art is in the ambitious goal of producing competitive interfaces out of abstract specifications that users can produce with reasonable effort.

4. Models in user interface development

The use of models and their automated transformation to executable UI source code are a promising approach to ease the process of UI development for several reasons. One reason is that modeling is on a higher level of abstraction than writing program code. This allows the designer to concentrate on high-level aspects of the interaction instead of low-level representation/programming details and supposedly makes modeling more affordable than writing program code. Another reason is that the difference in the level of abstraction makes models reusable and a suitable means for multi-platform applications, as one model can be transformed into several concrete implementations. This transformation is ideally even fully automatic. One further reason is that models, if automatically transformable, facilitate system modifications after the first development cycle. Changes on the requirements can be satisfied through changes on the models which are subsequently automatically propagated to the final UI through performing the transformations anew. A good overview of current state-of-the-art models, approaches and their use in the domain of UI development is given in Van den Bergh et al. (2010). It is notable that most approaches in the field of automated UI generation are based on the Model Driven Architecture[1] (MDA) paradigm. Such approaches use a set of models to capture the different aspects involved and apply model transformations while refining the input models to the source code for the final UI. In this section we will introduce and discuss model-driven UI development approaches that support the automated transformation of high-level interaction models to UI source code. We will highlight some of their strong points and shortcomings based on the criteria that we defined in section 3.

The primary focus of our criteria is the comparison of high-level models that are used as input for automated generation of user interfaces. Such models are typically tightly linked to a dedicated transformation approach to increase the *degree of operationalization* and the *adaptability in realization*. This tight coupling requires not only the comparison of the models, but also of the corresponding transformation approaches. We will use the Cameleon Reference

[1] `http://www.omg.org/mda/`

Framework by Calvary et al. (2003), a widely applied classification scheme for models used in UI generation processes, to determine the level of abstraction for the models to compare. The Cameleon Reference Framework defines four different levels of abstraction. These levels are from abstract to concrete:

1. *Tasks & Concepts*. This level accommodates high-level interaction specifications.
2. *Abstract UI*. This level accommodates a modality and toolkit-independent UI specification.
3. *Concrete UI*. This level accommodates a modality-dependent but still toolkit-independent UI specification.
4. *Final UI*. This level accommodates the final source code representation of the UI.

We apply our criteria to models on the tasks & concepts level and their transformation approaches.

Let us introduce a small excerpt from a flight booking scenario, which we will use to illustrate the presented approaches. First, the *System* asks the *User* to select a departure and a destination airport. Next the System provides a list of flights between the selected airports to the User. The User selects a flight and the System checks whether there are seats available on this flight or not (i.e., already overbooked). Finally, the System either asks the User to select a seat or informs him that the flight is already overbooked.

4.1 Discourse-based Communication Models

Discourse-based *Communication Models* provide a powerful means to specify the interaction between two parties on the tasks & concepts level. They integrate three different models to capture the aspects required for automated transformations (i.e., source code generation). Communication Models use a *Domain-of-Discourse Model* to capture the required aspects of the application domain. Moreover, they use an *Action-Notification Model* to specify actions that can be performed by either of the interacting parties and notifications that can be exchanged between them. The core part of the Communication Model is the *Discourse Model* that models the flow of interaction between two parties as well as the exchanged information (i.e., message content). The Discourse Model is based on human language theories and provides an intuitive way for interaction designers to specify the interaction between a user and a system. Discourse Models use Communicative Acts as basic communication units and relate them to capture the flow of interaction. The Communicative Acts are based on Speech Acts as introduced by Searle (1969). Typical turn takings like question-answer are modeled through Adjacency Pairs, derived from Conversation Analysis by Luff et al. (1990). Rhetorical Structure Theory (RST) by Mann & Thompson (1988) together with Procedural Relations are used to relate the Adjacency Pairs and provide the means to capture more complex flows of interaction. Discourse Models specify two interaction parties. Each Communicative Act is assigned to one of the two interacting parties and specifies the content of the exchanged messaged via its *Propositional Content*. The Propositional Content refers to concepts specified in the Domain-of-Discourse and the Action-Notification Model and is important for the operationalization of Communication Models (see Popp & Raneburger (2011) for details). Thus, the Discourse, the Domain-of-Discourse and the Action-Notification Model form the Communication Model which provides the basis for automated source code generation.

Let us use our small flight selection scenario to illustrate the discourse-based approach. Figure 1 shows the graphical representation of the Discourse Model for our scenario. This Discourse Model defines two interaction parties - the Customer (green or dark) and the

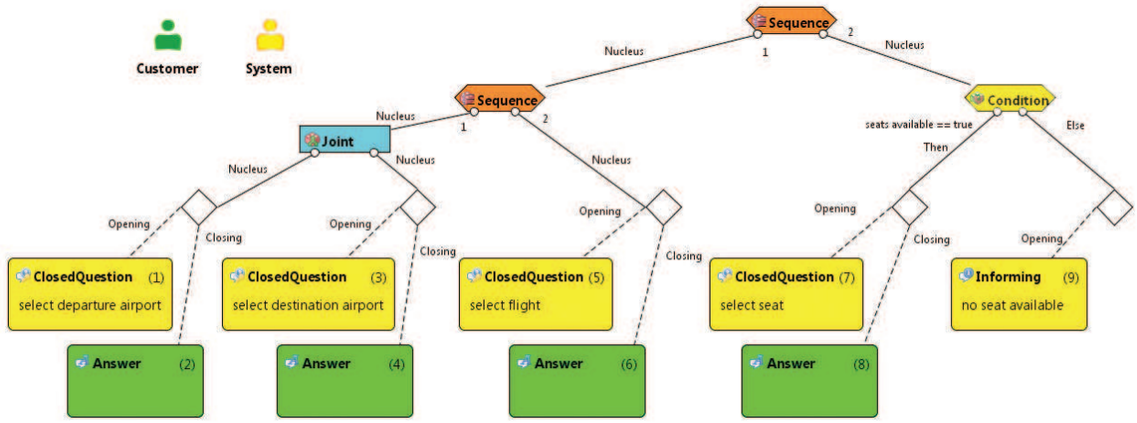

Fig. 1. Flight Booking Discourse Model from Raneburger, Popp, Kaindl & Falb (2011)

System (yellow or light). The Communicative Acts that are exchanged are represented by rounded boxes and the corresponding Adjacency Pairs by diamonds. The Adjacency Pairs are connected via RST or Procedural Relations. The green (or dark) and yellow (or light) fill color of the elements indicates the assigned interaction party.

Ease of Use — A graphical representation of Discourse Models eases their use for the designer. Various tutorials indicate that Discourse Models are intuitive to use during an informal design phase due to their human language theory basis. They support easy modeling of typical turn-takings in a conversation through the Adjacency Pairs and the specification of a more complex interaction through their Relations.

A high degree of operationalization for Communication Models is provided by the Unified Communication Platform (UCP) and the corresponding UI generation framework (UCP:UI). The aim during the development of UCP and UCP:UI was to stay compliant or apply well-established specification techniques so that only limited teaching is required. Therefore, an SQL-like syntax is used to specify the Propositional Content of each Communicative Act. Cascading Style Sheets[2] are used for style and layout specifications.

Degree of Operationalization — Discourse-based Communication Models can be operationalized with UCP and UCP:UI. A high degree of operationalization, however, requires more detailed specifications in the input models. Communication Models use the Propositional Content of each Communicative Act and the additional specification of conditions for Relations to provide the needed information for their operationalization and to specify the interface between UI and application logic. The Propositional Content specifies the content of the exchanged messages (i.e., Communicative Acts) and how they shall be processed by the corresponding interaction party. Popp & Raneburger (2011) show that the Propositional Content provides an unambiguous specification of the interface between the two interacting agents. In case of UI generation, the Propositional Content specifies the references to business logic components.

Additionally to the Propositional Content, Popp et al. (2009) include UML-state machines[3] in UCP to clearly define the procedural semantics of each Discourse Model element. Hence, each Discourse Model can be mapped to a finite-state machine. This composite state machine

[2] http://www.w3.org/Style/CSS/
[3] http://uml.org

is used to derive and define the corresponding UI behavior in case of UI generation (see Raneburger, Popp, Kaindl & Falb (2011)).

The runtime environment uses a Service-oriented Architecture and is provided by UCP Popp (2009). Figure 2 illustrates the operationalization of the Communication Model. The upper part depicts the integration of the Discourse, the Domain-of-Discourse and the Action-Notification Model into the Communication Model. The lower part shows that the Communication Model provides an interface that supports the distribution of the application and the generated UI on different machines. The *System* and the *Customer* communicate through the exchange of Communicative Acts over the Internet.

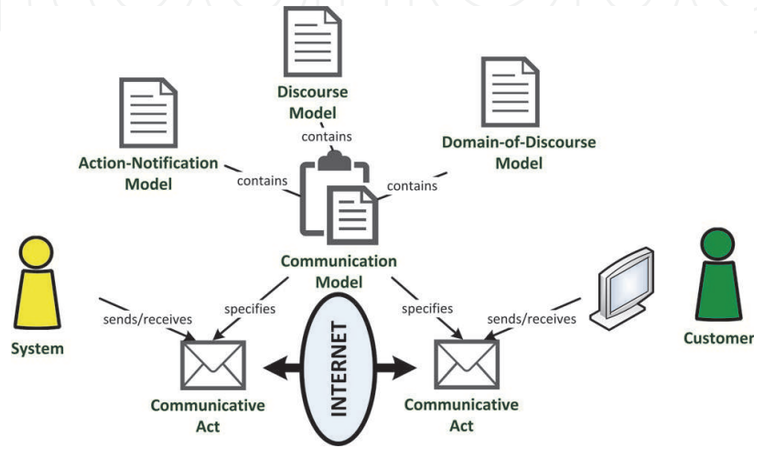

Fig. 2. The Communication Model as Runtime Interface

Coverage — Discourse Models define two abstract interaction parties. This makes them suitable to model not only human-machine but also machine-machine interaction as stated by Falb et al. (2006). Interaction Parties can be assigned to Communicative Acts as well as to Relations. Therefore, Communication Models provide a means to explicitly specify the interaction party on which the progress of the interaction depends at a certain time.

As mentioned above, each Propositional Content is defined for a certain Communicative Act, which form the basic communication units. This implies that Communicative Acts and their corresponding values cannot be updated after they have been sent to the other interaction party. For example, let's consider the selection of a departure and a destination airport in a flight selection scenario. It would be sensible to limit the list of destination airports according to the selected departure airport. If the selection of both airports is concurrently available this cannot be done, because no Communicative Acts are exchanged between the UI and the business logic between the selection.

Adaptability in Realization — Discourse-based Communication Models are device- and platform-independent. For a device-specific UI generation however, additional information about the target device, style and layout must be provided. UCP provides this information in form of default templates that can be selected and modified by the designer.

UCP:UI incorporates a methodology to transform Communication Models into WIMP-UIs for different devices and platforms at compile time. It uses automated optimization to generate UIs for different devices as presented in Raneburger, Popp, Kavaldjian, Kaindl & Falb (2011). Because of this optimization there is no user interface model on abstract UI level. However, we create a consistent screen-based UI representation on concrete UI level — the Screen Model.

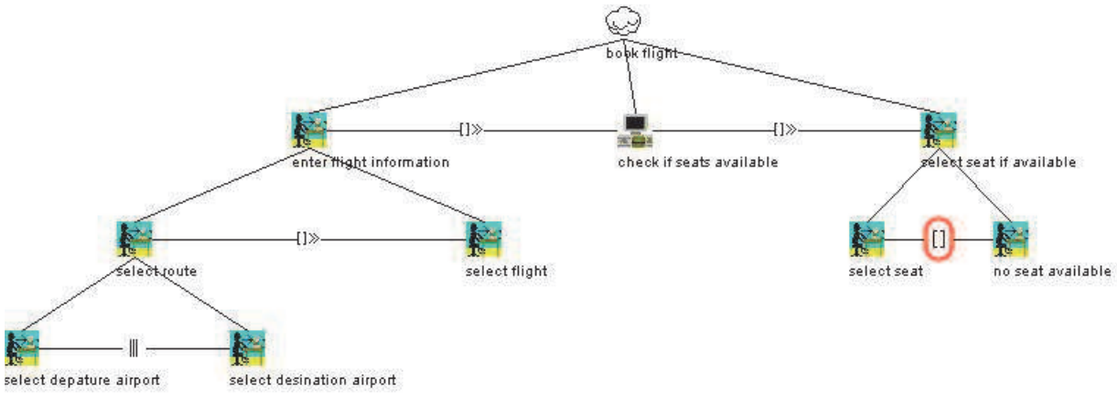

Fig. 3. Flight Booking Concur Task Tree Model

Raneburger (2010) argues that the adaptability during the UI generation process is important in order to generate a satisfying UI for the end user. This is due to the reason that high-level models for UI generation per se do not provide the appropriate means to specify non-functional requirements like layout or style issues. UCP:UI provides the possibility to specify layout and style issues either in the transformation rules used to transform the Communication Model into a Structural Screen Model, or via CSS.

Reuse of Partial Specification — So far there is no support for reuse of partial specifications.

4.2 Task models

Task models provide designers with a means to model a user's tasks to reach a specific goal. A thorough review of task models can be found in Limbourg & Vanderdonckt (2003) and a taxonomy for the comparison of task models has been developed by Meixner & Seissler (2011). In our chapter we focus on task models using the Concur Task Tree (CTT) notation as defined by Paternò et al. (1997). This notation is the de-facto standard today.

Each CTT model specifies its goal as an abstract root task. In order to achieve this goal the root task is decomposed into sub-tasks during the model creation phase. The leaf nodes of the CTT model are concrete *User*, *Interaction* or *Machine Tasks*. The subtasks on each level are related through *Temporal Operators*. These operators are used to specify the order in which the tasks have to be performed to reach the specific goal.

Figure 3 depicts the CTT Model for our running example. The abstract root task *bookflight* is decomposed into several concrete Interaction or Machine tasks that are required to reach the specific goal (i.e., to select a flight ticket). These concrete tasks are either performed by a human user (Interaction Tasks) or the system (Machine Tasks). Interaction Tasks are depicted as a human user in front of a computer and Machine Tasks as a small computer. Tasks on the same level in a CTT diagram are related via a Temporal Operator. The tasks *select departure airport* and *select destination airport* are on the same level and shall be enabled at the same time. This is expressed by the *interleaving* Temporal Operator that relates them. The *select flight* task requires the information of the airports selected in the *select route* task. Therefore, the *enabling with information passing* Temporal Operator is used to relate these tasks. Our scenario states that the machine shall check whether seats are available or not after a certain flight has been selected (i.e., after the *enter flight information* task is finished) and either offer a list of seats or

inform the user that no seats are available. We modeled this decision with the *choice* Temporal Operator that relates the *select seat* and *no seat available* interaction tasks.

Task Models, just like Communication Models, have been designed in order to support automated UI generation. In this chapter we use our criteria to compare two major task-based UI development frameworks: the MARIA-Environment (MARIAE) and the USer Interface eXtensible Markup Language[4] (UsiXML) framework.

MARIAE is based on the MARIA user interface specification language developed by Paternò et al. (2009) and provides tool-based design support on all four levels of the Cameleon Reference Framework.

The UsiXML language forms the basis of the UI development framework presented in Vanderdonckt (2008). UsiXML is a XML-compliant markup language that describes the UI for multiple contexts of use such as Character User Interfaces (CUIs), Graphical User Interfaces (GUIs), Auditory User Interfaces, and Multimodal User Interfaces. The UsiXML framework uses CTT models on the tasks & concepts level and supports the designer with tools during the UI generation. The interoperability between the tools is accomplished through the common use of UsiXML. The focus of UsiXML development team is not the development of UI models and a generation approach but the creation of a UI specification language that supports the specification of all needed models with one language.

Ease of Use — A graphical representation for all CTT elements together with tool support through the CTT-Environment (CTTE) developed by Mori et al. (2002) makes the creation of task models affordable for designers. MARIAE as well as the UsiXML framework provide tool support on all four levels of the Cameleon Reference Framework.

Degree of Operationalization — The degree of operationalization for fully specified task models is high. Both approaches use device-specific, but platform and modality independent CTT models on the tasks & concepts level. They provide tool support for the transformation into UIs for multiple modalities.

References to the application logic can be specified through the definition of modified objects for each task or Web service calls. However, it faces the same UI update problem as Communication Model.

Coverage — Task models are primarily used to model user-driven applications. User-driven in this context means that the user decides which tasks to execute next and not the system. CTT models in principle support the specification of preconditions in order to model scenarios in which the system decides what task to execute next. CTT does not support the unambiguous specification of such preconditions. Therefore, these preconditions are not considered during the course of the UI generation in MARIAE. This leads to the derivation of wrong Presentation Task Sets and the corresponding Navigators in the end and poses therefore a limitation of the coverage. The consideration of the following two aspects would solve this problem. First, a clear syntax for the specification of the preconditions is required. Second, these preconditions must be considered and evaluated during the UI generation process.

Figure 3 shows the CTT model that we created after having failed using the preconditions. The *choice* Temporal Operator, marked with a red (or gray) ellipse, represents the check for available seats. Figure 3 is not a true model of our scenario as CTT does not specify the user and the system explicitly as roles. Therefore, it does not support the assignment of

[4] `http://www.usixml.org/`

Temporal Operators to the machine or the system. Even our small scenario shows that there is an expressiveness problem if CTT models shall be used to model machine decisions. This problem could be solved if it would be possible to assign roles and specify conditions for Temporal Operators.

Apart from specifying the interaction between a user and a system, CTT models can also be used to specify the collaborative interaction between more than two users. Such Cooperative Task Models define an arbitrary number of interaction parties and the flow of interaction between them. The interaction between each interaction party and the system is specified through CTT models.

Adaptability in Realization — CTT models are device-specific. However, both approaches provide tools to adapt the CTT for different devices and contexts of use and to generate the corresponding UI at design time. Based on these UIs, both frameworks support the migration of an application's UI through migratory UIs (see Bandelloni & Paternò (2004) and Collignon et al. (2008)) that adapt to various contexts of use (i.e., device, environment, etc.) during runtime.

Reuse of Partial Specifications — To the best of our knowledge there is no support for reuse of partial specifications so far.

4.3 Models in the OO-Method

The *OO-Method* has been developed by Pastor et al. (2008) and introduces a so-called *Conceptual Model* to define all aspects that are needed to generate information system applications. The Conceptual Model consists of an *Object Model*, a *Dynamic Model*, a *Functional Model* and a *Presentation Model*. This method uses a model compiler to transform these four models into a UI for an application fully automatically. CTT models are only used on Computational Independent Level. They are not processed fully automatically, but rather define a basis for the creation of the four models mentioned above.

The OO-method has been tailored for the creation of information system applications. Therefore, it is *not easy to use* for untrained users. Furthermore, its focus on information systems limits the *coverage* of the corresponding models on the one hand, but increases their *degree of operationalization* on the other hand. Tool support for the OO-Method on an industrial scale is provided by the Olivanova transformation engine[5]. *Adaptability* for the resulting UI is considered as important and constitutes a current field of research (see Pederiva et al. (2007)). Their current approach is to provide additional rendering information in so called transformation templates developed by Aquino et al. (2008). This approach has been chosen as not all parts of the rendering engines and the corresponding models are accessible for alterations. To the best of our knowledge there is no support for reuse of partial specifications so far.

5. Assessment

In this section, we compare the models introduced in the previous section, according to the criteria defined before. Since neither their state of elaboration, nor the details of documentation available are such that an in depth comparison appears to be sensible, we summarize some of the criteria in our comparison. We characterize the state of all models

[5] `http://www.care-t.com`

with respect to single or related sets of criteria, and we contrast discourse-based with task models respectively OO models where appropriate.

Concerning the ease of use, there is sufficient evidence that both models behave reasonable. The discourse-based model has been presented in various tutorials, and participants were able to build simple models already after short training. Tasks models are well known and commonly used, which makes it plausible that they are even easier to use than the discourse-based model, since no training is required to get acquainted with idiosyncrasies of the model. Both models support the user in building models according to syntactic conventions, but they fail to provide means of repair semantic errors in user specifications. Graphical CTT models use a different icon for each temporal operator that represents its meaning. Compared to Communication Models such operators are more easy to read. However, RST-based Communication Model Relation provide additional semantic information that can be exploited to derive the layout of a resulting graphical UI or the emphasis of different parts in a speech UI. The OO-Method uses task models only during an informal design phase. The creation of the Conceptual Model requires detailed knowledge of the models involved. Therefore, such a model cannot be created by untrained users. Altogether, concretely assessing the ease of use depends on the particular user. If you are familiar with established modeling concepts the use of all models will be affordable, the Discourse Models even with less amount of training due to their natural language basis.

The operationalization is quite differently organized across models and approaches. In the discourse-based model, the abstract user specifications are operationalized and, by the aid of schematic patterns for rendering purposes, successively transduced into executable code. For the task model, operationalization depends on a suitable orchestration of transformation processes that mediate between the layers of representation. A weak point in both models is the reference to business logic. While the discourse-based model offers an admittedly rudimentary way to handle references to business logic elements, this part seems not well supported formally in the task model. The OO-Method focuses on the creation of UIs for information systems. Information systems require only a limited set of business logic functionality. The OO-Method's Dynamic Model together with its Functional Model provide an explicit specification of the objects managed by the business logic and the events changing their states.

Discourse Models and CTT Models are both on the tasks & concepts level of the Cameleon Reference Framework. One could argue however that Discourse Models are on a slightly higher level of abstraction as their Relations introduce an additional semantic layer and are decoupled from their procedural semantics through an explicit state machine representation. Hence, Discourse Models have a greater coverage but per se a lesser degree of operationalization than CTT models.

The coverage can be assessed easier for the discourse-based model, since its building blocks have close relations to the categories of coverage, as they appear in our list of criteria. This model can handle quite well various sorts of conditionally determined discourse continuations, as well as groupings of semantically-related discourse acts. Simultaneous actions, though in principle expressible, are not yet fully supported by the operationalization method. Finally, advanced discourse continuations, that is, topic changes involving the implicit leave of open subdialogs is not elaborated yet. Assessing the coverage for task models is a bit speculative, since this cannot be done on the level of tasks per se. It depends on how the task model specifications are mapped onto compositions of interactions, by transformations between these layers of representation. This transformation is quite challenging, including a

Criterion	Discourse-based	Task-based	OO-method
Ease of use	reasonable, some experimental evidence	best known approach	detailed specifications needed
Operational-ization	systematic process, clear application logic interface	good tool support on all abstraction levels	direct model compilation
Coverage	good repertoire, but no advanced discourse continuation patterns	user-driven applications	tailored to information systems
Adaptation	explicit support, some degree of elaboration	device specific input model	device specific input model
Reuse	not yet developed	not yet developed	not yet developed

Table 1. Comparing Discourse-based, task-based and OO approach

variety of choices with substantial differences in effectiveness. Intuitively, the coverage of the OO-Method seems smaller as its focus is on information systems only.

The attitude towards adaptation is quite divergent between the competing approaches. While discourse-based models explicitly support the adaptation to multiple devices from the same abstract representation, the task model requires the designer to take into account properties of the intended device already from the beginning. Thus, this part of the philosophy behind the task model makes the generation process a bit easier, but it may turn out to be quite awkward, if changes of the intended device or extension in their variation prove necessary or at least desirable. Elaborations of the discourse-based model have already demonstrated some success in producing structural variations driven by device constraints from the same abstract specifications, it remains to be seen how far the repertoire of device variants and associated rendering alternations can be extended. The OO-Method uses a device-specific Presentation Model and does not support adaptability for different devices during its compilation process.

Concerning the last category of criteria, reuse, it is not surprising that neither of the two approaches has to offer something yet. Reuse of partial specifications is quite an advanced issue in the context of HCI, since it is a priori not clear how portions suitable for reuse can be precisely defined, and how they need to be adapted to the context in which they are reused. For the design of larger interfaces, however, such a feature will eventually be indispensable.

Major differences between the models are summarized in Table 1. Some of the major problems are shared by all approaches: missing user support for handling semantic errors, reference to business logic elements, and reuse of partial specifications.

Altogether, the competing models turn out to exhibit complementary properties in several factors, which has its source in fundamental differences of the underlying philosophy. The task model treats the interface design as an integral part of the overall software design. Specifications to be made are decoupled into a primary, highly abstract design at the task level, and subsequent transformations, which gradually concretize the task model into a working interface. Success of this model widely depends on the suitability of the intermediate representation levels and on the skill of the designer in finding effective optimizations when building transformations mapping between adjacent representation levels. The discourse-based model has quite different priorities. The proper design resides on the level of discourse interactions, which is a level deeper than the primary design level of the task model. Consequently, the designer can concentrate his efforts on precisely this level, which makes his

task quite uniform. It is assumed that he is able to structure the design of the interaction specifications in a suitable manner, without an abstracting task model, but supported by the underlying linguistic concept. Moreover, user interface production and adaptation for one or several devices is automated, guided by declarative representations of device constraints.

Table 1 indicates that each modeling and transformation approach has its own limitations. Therefore, it is important to have a set of criteria as provided in our chapter, to compare them in order to find the most appropriate model and approach for a given problem.

6. Conclusion and future work

In this paper, we have described and compared three models for HCI design which operate on some abstract, semantically-oriented levels - a discourse-based, a task model, and an OO model. We have made this comparison along an advanced set of criteria, which has demonstrated achievements and shortcomings of these approaches, but also complementary strengths and weaknesses grounded in the different nature of these approaches.

When expanding the coverage in these models, difficulties are expected to be complementary, according to the differences in the architectural design of the models. In the discourse-based model, additional representation elements must be defined to enable the user to built specifications for more advanced discourse situations. Since these elements are likely to be associated with relatively complex semantics, similar to the procedural relations, much care must be devoted to this task - users must get a handle on understanding how to use these elements, in order to achieve a desired system behavior. Moreover, modules responsible for operationalization must be enhanced accordingly, which may be challenging for some complex representation elements. In contrast to that, additional representation elements in task and OO models probably need not to be semantically complex, but several such elements from different representation levels are likely to contribute to specific coverage extensions. In such a setting, the challenge is to define complementing expressive means adequately. For the user, it is important to understand, on which level he needs to make partial specifications, and how they interact to obtain a desired system behavior.

In order to strengthen these models, they should address several factors that became apparent through our comparison: the discourse-based model may profit from some sort of relations between discourse fragments and tasks, inspired by the task model, but different from the use there. The task model may allow for some degrees of adaptation to device variants, although not in the principled manner as the discourse-based model does. The OO model may adapt some of the information encapsulated in discourse relations to support adaptation in the rendering process, and it may put some emphasis on making the task of the designer less dependent on knowledge and training. Finally, all models should take the incorporation to business logic more serious, and try to address some more advanced and effective patterns of communication as well as measure to support some degree of reuse of partial specifications.

7. References

Aquino, N., Vanderdonckt, J., Valverde, F. & Pastor, O. (2008). Using profiles to support transformations in the model-driven development of user interfaces, *Proceedings of the 7th International Conference on Computer-Aided Design of User Interfaces (CADUI 2008)*, Springer.

Bandelloni, R. & Paternò, F. (2004). Migratory user interfaces able to adapt to various interaction platforms, *International Journal of Human-Computer Studies* 60(5-6): pp. 621–639. HCI Issues in Mobile Computing.

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L. & Vanderdonckt, J. (2003). A unifying reference framework for multi-target user interfaces, *Interacting with Computers* 15(3): pp. 289–308. Computer-Aided Design of User Interface.
URL: *http://www.sciencedirect.com/science/article/pii/S0953543803000109*

Collignon, B., Vanderdonckt, J. & Calvary, G. (2008). Model-driven engineering of multi-target plastic user interfaces, *Proceedings of the Fourth International Conference on Autonomic and Autonomous Systems (ICAS 2008)*, IEEE Computer Society, Washington, DC, USA, pp. 7–14.

Falb, J., Kaindl, H., Horacek, H., Bogdan, C., Popp, R. & Arnautovic, E. (2006). A discourse model for interaction design based on theories of human communication, *Extended Abstracts on Human Factors in Computing Systems (CHI '06)*, ACM Press: New York, NY, pp. 754–759.

Heeman, P. (2007). Combining reinforcement learning with information-state update rules, *Proceedings of the North American Chapter of the Association for Computational Linguistics Annual Meeting*, pp. 268–275.

Kreutel, J. & Matheson, C. (2003). Incremental information state updates in an obligation-driven dialogue model, *Logic Journal of the IGPL* 11(4): pp. 485–511.

Limbourg, Q. & Vanderdonckt, J. (2003). Comparing task models for user interface design, *in* D. Diaper & N. Stanton (eds), *The Handbook of Task Analysis for Human-Computer Interaction*, Lawrence Erlbaum Associates, Mahwah, NJ, USA, chapter 6.

Luff, P., Frohlich, D. & Gilbert, N. (1990). *Computers and Conversation*, Academic Press, London, UK.

Mann, W. C. & Thompson, S. (1988). Rhetorical Structure Theory: Toward a functional theory of text organization, *Text* 8(3): pp. 243–281.

Matheson, C., Poesio, M. & Traum, D. (2000). Modelling grounding and discourse obligations using update rules, *Proceedings of the 1st Annual Meeting of the North American Association for Computational Linguistics (NAACL2000)*, pp. 1–8.

Meixner, G. & Seissler, M. (2011). Selecting the right task model for model-based user interface development, *ACHI 2011, The Fourth International Conference on Advances in Computer-Human Interactions*, pp. 5–11.

Meteer, M. (1992). *Expressibility and the problem of efficient text planning*, St. Martin's Press, Inc. New York, NY, USA.

Mori, G., Paternò, F. & Santoro, C. (2002). Ctte: Support for developing and analyzing task models for interactive system design, *IEEE Transactions on Software Engineering* 28: pp. 797–813.

Pastor, O., España, S., Panach, J. I. & Aquino, N. (2008). Model-driven development, *Informatik Spektrum* 31(5): pp. 394–407.

Paternò, F., Mancini, C. & Meniconi, S. (1997). ConcurTaskTrees: A diagrammatic notation for specifying task models, *Proceedings of the IFIP TC13 Sixth International Conference on Human-Computer Interaction*, pp. 362–369.

Paternò, F., Santoro, C. & Spano, L. D. (2009). Maria: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments, *ACM Trans. Comput.-Hum. Interact.* 16: pp. 19:1–19:30.
URL: *http://doi.acm.org/10.1145/1614390.1614394*

Pederiva, I., Vanderdonckt, J., España, S., Panach, I. & Pastor, O. (2007). The beautification process in model-driven engineering of user interfaces, *Proceedings of the 11th IFIP TC*

13 International Conference on Human-Computer Interaction — INTERACT 2007, Part I, LNCS 4662, Springer Berlin / Heidelberg, Rio de Janeiro, Brazil, pp. 411–425.
URL: *http://dx.doi.org/10.1007/978-3-540-74796-3_39*

Popp, R. (2009). Defining communication in SOA based on discourse models, *Proceeding of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications (OOPSLA '09)*, ACM Press: New York, NY, pp. 829–830.

Popp, R., Falb, J., Arnautovic, E., Kaindl, H., Kavaldjian, S., Ertl, D., Horacek, H. & Bogdan, C. (2009). Automatic generation of the behavior of a user interface from a high-level discourse model, *Proceedings of the 42nd Annual Hawaii International Conference on System Sciences (HICSS-42)*, IEEE Computer Society Press, Piscataway, NJ, USA.

Popp, R. & Raneburger, D. (2011). A high-level agent interaction protocol based on a communication ontology, *in* C. Huemer & T. Setzer (eds), *EC-Web 2011*, Vol. 85 of *Lecture Notes in Business Information Processing*, Springer Berlin Heidelberg, pp. 233–245.

Raneburger, D. (2010). Interactive model driven graphical user interface generation, *Proceedings of the 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '10)*, ACM, New York, NY, USA, pp. 321–324.
URL: *http://doi.acm.org/10.1145/1822018.1822071*

Raneburger, D., Popp, R., Kaindl, H. & Falb, J. (2011). Automated WIMP-UI behavior generation: Parallelism and granularity of communication units, *Proceedings of the 2011 IEEE International Conference on Systems, Man and Cybernetics (SMC 2011)*.

Raneburger, D., Popp, R., Kavaldjian, S., Kaindl, H. & Falb, J. (2011). Optimized GUI generation for small screens, *in* H. Hussmann, G. Meixner & D. Zuehlke (eds), *Model-Driven Development of Advanced User Interfaces*, Vol. 340 of *Studies in Computational Intelligence*, Springer Berlin / Heidelberg, pp. 107–122.
URL: *http://dx.doi.org/10.1007/978-3-642-14562-9_6*

Reiter, E. (1994). Has a consensus nl generation architecture appeared, and is it psycholinguistically plausible?, *Proceeding INLG '94 Proceedings of the Seventh International Workshop on Natural Language Generation*, Association for Computational Linguistics.

Searle, J. R. (1969). *Speech Acts: An Essay in the Philosophy of Language*, Cambridge University Press, Cambridge, England.

Traum, D. & Larsson, S. (2003). The information state approach to dialogue management, *in* R. Smith & J. van Kuppevelt (eds), *Current and New Directions in Discourse and Dialogue*, Kluwer Academic Publishers, Dordrecht, pp. 325–353.

Van den Bergh, J., Meixner, G., Breiner, K., Pleuss, A., Sauer, S. & Hussmann, H. (2010). Model-driven development of advanced user interfaces, *Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems*, CHI EA '10, ACM, New York, NY, USA, pp. 4429–4432.
URL: *http://doi.acm.org/10.1145/1753846.1754166*

Vanderdonckt, J. M. (2008). Model-driven engineering of user interfaces: Promises, successes, and failures, *Proceedings of 5th Annual Romanian Conf. on Human-Computer Interaction*, Matrix ROM, Bucaresti, pp. 1–10.

Human Machine Interaction - Getting Closer

Edited by Mr Inaki Maurtua

ISBN 978-953-307-890-8
Hard cover, 260 pages
Publisher InTech
Published online 25, January, 2012
Published in print edition January, 2012

In this book, the reader will find a set of papers divided into two sections. The first section presents different proposals focused on the human-machine interaction development process. The second section is devoted to different aspects of interaction, with a special emphasis on the physical interaction.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Helmut Horacek, Roman Popp and David Raneburger (2012). Automated Generation of User Interfaces – A Comparison of Models and Future Prospects, Human Machine Interaction - Getting Closer, Mr Inaki Maurtua (Ed.), ISBN: 978-953-307-890-8, InTech, Available from: http://www.intechopen.com/books/human-machine-interaction-getting-closer/automated-generation-of-user-interfaces-a-comparison-of-models-and-future-prospects

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen